Habermann, P.; Chi, C. C.; Alvarez-Mesa, M. & Juurlink, B.

Application-Specific Cache and Prefetching for HEVC CABAC Decoding

Habermann, P., Chi, C. C., Alvarez-Mesa, M., & Juurlink, B. (2017). Application-Specific Cache and Prefetching for HEVC CABAC Decoding. IEEE MultiMedia, 24(1), 72–85.
https://doi.org/10.1109/mmul.2017.12

Application-Specific Cache and Prefetching for HEVC CABAC Decoding

Habermann, Philipp; Chi, Chi Ching; Alvarez-Mesa, Mauricio; Juurlink, Ben

Technische Universität Berlin

Abstract:

Context-based Adaptive Binary Arithmetic Coding (CABAC) is the entropy coding module in the HEVC/H.265 video coding standard. As in its predecessor, H.264/AVC, CABAC is a well-known throughput bottleneck due to its strong data dependencies. Besides other optimizations, the replacement of the context model memory by a smaller cache has been proposed for hardware decoders, resulting in an improved clock frequency. However, the effect of potential cache misses has not been properly evaluated. This work fills the gap by performing an extensive evaluation of different cache configurations. Furthermore, it demonstrates that application-specific context model prefetching can effectively reduce the miss rate and increase the overall performance. The best results are achieved with two cache lines consisting of four or eight context models. The 2 × 8 cache allows a performance improvement of 13.2 percent to 16.7 percent compared to a non-cached decoder due to a 17 percent higher clock frequency and highly effective prefetching. The proposed HEVC/H.265 CABAC decoder allows the decoding of high-quality Full HD videos in real-time using few hardware resources on a low-power FPGA.

High Efficiency Video Coding (HEVC/H.265; here, *HEVC*) is the most recent video coding standard developed by the Joint Collaborative Team on Video Coding (JCT-VC).[1] HEVC allows video compression with the same perceptive quality as its predecessor H.264/AVC[2] (here, *H.264*) while requiring only half the bitrate. Due to its sequential nature, Context-based Adaptive Binary Arithmetic Coding (CABAC)[3,4] was a throughput bottleneck in H.264 decoding. This is still the case in HEVC. High-throughput requirements must be fulfilled to achieve real-time decoding of high-quality, high-resolution videos. This is a challenging task, especially on mobile devices and other power- and performance-constrained systems, making energy-efficient hardware solutions necessary.

To address this, in addition to many other optimizations, researchers have proposed replacing the context model memory in the data path with a smaller cache.[5,6] The goal is to shorten the critical path and increase the clock frequency and throughput. Unfortunately, the effect of potential cache misses has not been properly evaluated. Cache misses result in a performance degradation that might jeopardize the throughput improvements achieved by the cache's introduction. Although Yu Hong and colleagues proposed prefetching to address this issue,[7] they did not quantitatively evaluate their proposal. Here, we evaluate caches and prefetching to see if they are gainful optimizations.

Our work makes three contributions to HEVC CABAC hardware decoding by offering

- an optimized context model cache architecture (resulting from our extensive evaluation of different configurations);
- an efficient context model memory layout optimized for spatial locality and prefetching efficiency, as well as the corresponding adaptive prefetching algorithm; and
- an evaluation of prefetching efficiency, realtime decoding capabilities, and energy efficiency for different cache configurations.

Here, we offer background on CABAC decoding, describe the proposed decoder architecture, and discuss evaluations of it in terms of the cache miss rate and prefetching efficiency, realtime capabilities, hardware costs, and energy efficiency.

HEVC CABAC Decoding

CABAC is the entropy coding module of the HEVC standard. Following is the theoretical background required to follow our argument here; an overview of other proposed optimizations for HEVC CABAC hardware decoding is covered in the “Related Work” sidebar.

In the HEVC encoder, CABAC is responsible for the efficient compression of syntax elements during the bitstream generation. The CABAC decoder performs the inverse process to restore the compressed syntax elements from the bitstream. Syntax elements are a compact representation of a frame, achieving a higher compression rate than is possible with a sample-based representation. An example syntax element is the pred.mode.flaq, which indicates whether a block of samples is predicted from a block in a previous frame or an adjacent block in the same frame. The difference between the original block and the predicted block is the residual. HEVC uses transform blocks to signal a transformed version of the residual. Because we refer to these transform block syntax elements frequently, we now offer a brief overview.

A 4×4 transform block contains 16 integer transform coefficients. First, a significance map indicates the nonzoro coefficients. Therefore, the x- and y-components of the last significant coefficient's position in scan order are signaled with the last_sig_coeff_x/y_prefix and last_sig_coeff_x/y_suffix syntax elements. Additionally, a sig_coeff_flag is used for every coefficient before the last significant coefficient to indicate if it is also significant (nonzero).

Next, the significant coefficients' level must be determined. For that, coeff_abs_level_greater1_flag, coeff_abs_level_greater2_flag, and coeff_abs_level_remaining syntax elements are used to successively increment the absolute coefficient level to the required value. Finally, a coeff_sign_flag is assigned to every nonzero coefficient.

Syntax elements consist of one or more binary symbols (bins), because this is required by the binary arithmetic coder. Context-coded bins are associated with context models to exploit statistical properties and thereby increase the compression rate. A context model is a variable containing the estimated probability that a bin has a specific value – that is, the most probable

symbol (MPS). After the decoding of each context-coded bin, the probability of the corresponding context model is updated to give a better estimation the next time it is used. For some bins, the probabilities cannot be accurately predicted; these so-called *bypass-coded bins* are thus decoded without context models.

Related Work in Cabac Hardware Decoding Optimizations

Many hardware decoder implementations for Context-based Adaptive Binary Arithmetic Coding (CABAC) have been proposed. Although most of them target the H.264/ AVC video coding standard (here, *H.264*), the general ideas are also applicable to the more recent HEVC/H.265 (here, *HEVC*), because the entropy coding is very similar in both standards. While the arithmetic coding engines and context model update procedures are exactly the same, differentsyntax elements are used in HEVC to represent the block structure, intra prediction mode, motion data, transform coefficients, and parameters of the novel sample adaptive offset filter. Additionally, HEVC CABAC is designed to allow higher throughput by various optimizations in the standard, such as a reduced fraction of context-coded bins, grouping of bypass-coded bins, and a reduction of the total number of bins, as well as more relaxed parsing and context selection dependencies.[1] Strong bin-to-bin dependencies make low-level parallelization of CABAC decoding very challenging. Although high-level parallelization – such as that in Chi Ching Chi and his colleagues' implementation[2] – is possible in HEVC, it must be enabled during encoding, which is not mandatory.

Two architectural optimizations have proven effective, and at least one of them can be found in almost every proposed CABAC hardware decoder. The first is *parallel decoding of multiple bins per clock cycle*, implemented by Yu-Hsin Chen and Vivienne Sze,[3] Chung-Hyo Kim and In-Cheol Park,[4] and Pin-Chih Lin and colleagues.[5] This optimization is well suited for bypass-coded bins, because their decoding process is simple and does not require context models. It is also very efficient in HEVC, because it can be used more frequently due to the grouping of bypass-coded bins. Furthermore, the group size is also often known before the decoding process, which removes parsing dependencies. On the other hand, parallel decoding of context-coded bins is more complex; either two different context models are needed, which requires multiport memories,[5] or the same context model is used twice and must be updated in between uses. Often, a concatenation of two arithmetic decoders is used, almost doubling the critical path's length. This can be improved when the second decoder speculates that the first decodes the most probable symbol (MPS).[4] In that case, the critical path is not significantly affected, but the second bin must be discarded if the speculation was wrong.

The second optimization is *pipelining*, which is used to overlap the decoding of consecutive bins and thereby increase the throughput. Most proposals agree on a conceptual four-stage pipeline for CABAC decoding: context selection, context load, arithmetic decoding, and debinarization. Neighboring stages are often merged, because efficiently implementing deep pipelining is very complex for CABAC due to strong bin-to-bin dependencies. For example, the context selection depends on the results of the arithmetic decoding and debinarization stages, which might lead to a flush of the complete pipeline. Decoders with three pipeline stages were presented in several research efforts.[4, 6, 7]

More efficient pipelining implementations are possible if dependencies are removed, because a bin can have only two different states. This enables the removal of dependencies from the decoded bin, because results can be computed for both possible states and the correct result is selected as soon as the decoded bin is available. This technique can be applied to the context selection, removing the dependency from the second last bin, as well as for debinarization, which can be performed in parallel with arithmetic decoding and thereby shorten the pipeline.

Chen and Sze[3] proposed an even deeper pipeline: A fifth pipeline stage is added at the beginning to compute a binary decision tree, which is used for state prefetching in the remaining stages, thereby reducing pipeline stalls. Their implementation also decodes up to two bypass-coded bins per cycle, resulting in a throughput of up to 1.7 Gbins/s, which represents the state of the art.

The context load stage can be shortened when the context model memory is replaced by a small cache. The stage might even be removed when the cache access fits into an adjacent stage. The shorter pipeline might result in fewer pipeline stalls. Cached designs have been proposed by several researchers,[6, 8, 9] but the potential performance degradation due to cache misses has not been evaluated. Although some researchers used prefetching to reduce the cache miss rate,[9] they did not provide results for this optimization. Our work evaluates both the cache miss rate and prefetching effectiveness. Furthermore, ours is the first HEVC CABAC hardware decoder with a context model cache.

We previously presented results of our work at the IEEE International Symposium on Multimedia (ISM 2015).[10] In this article, we expand those results to consider different cache line sizes; we have also improved the prefetching algorithm to adapt to dynamic video characteristics. Finally, we add results here regarding real-time decoding capabilities and power efficiency.

References

1. V. Sze and M. Budagavi, “High Throughput CABAC Entropy Coding in HEVC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1778–1791.
2. C.C. Chi , “Parallel Scalability and Efficiency of HEVC Parallelization Approaches,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1827–1838.
3. Y.-H. Chen and V. Sze, “A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-tier Applications,” IEEE Trans. Circuits and Systems for Video Technology, vol. 25, no. 5, 2015, p. pp. 856–868.
4. C.-H. Kim and I.-C. Park, “High Speed Decoding of Context-based Adaptive Binary Arithmetic Codes Using Most Probable Symbol Prediction,” in IEEE Int'l Symp. Circuits and Systems (ISCAS), 2006, pp. 1707–1710.
5. P.-C. Lin, T.-D. Chuang, and L.-G. Chen, “A Branch Selection Multi-symbol High Throughput CABAC Decoder Architecture for H.264/AVC,” in IEEE Int'l Symp. Circuits and Systems (ISCAS), 2009, pp. 365–368.
6. Y. Yi and I.-C. Park, “High-Speed H.264/AVC CABAC Decoding,” IEEE Trans. Circuits and Systems for Video Technology, vol. 17, no. 4, 2007, pp. 490–494.
7. Y.-T. Chang, “A Novel Pipeline Architecture for H.264/AVC CABAC Decoder,” IEEE Asia Pacific Conf. Circuits and Systems (APCCAS), 2008, pp. 308–311.
8. Y.-C. Yang and J.-I. Guo, “High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications,” IEEE Trans. Circuits and Systems for Video Technology, vol. 19, no. 9, 2009, pp. 1395–1399.
9. Y. Hong , “A 360 Mbin/s CABAC Decoder for H.264/AVC Level 5.1 Applications,” in Proc. IEEE Int'l SoC Design Conf. (ISOCC), 2009, pp. 71–74.
10. P. Habermann , “Optimizing HEVC CABAC Decoding with a Context Model Cache and Application-specific Prefetching,” in Proc. 11 th IEEE Int'l Symp. Multimedia (ISM), 2015, pp. 429–434.

Architecture

We implemented the proposed decoder architecture with a two-stage pipeline (see Figure 1). We avoid the use of more pipeline stages to keep the design complexity low. The *context model selection* stage computes the decoder control state machine's next state, calculates the required context model's index, and accesses the context model cache. The cache contains context model sets (CMSs) and is clocked with a phase-shifted clock signal to allow access at the end of the pipeline stage. In the *binary arithmetic decoding* stage, one context-coded bin or up to two bypass-coded bins are decoded by the arithmetic decoder. The decoded bins are fed back to the first stage and the context model is updated and written back to the cache. Finally, debinarization is performed to build syntax elements from the decoded bins.

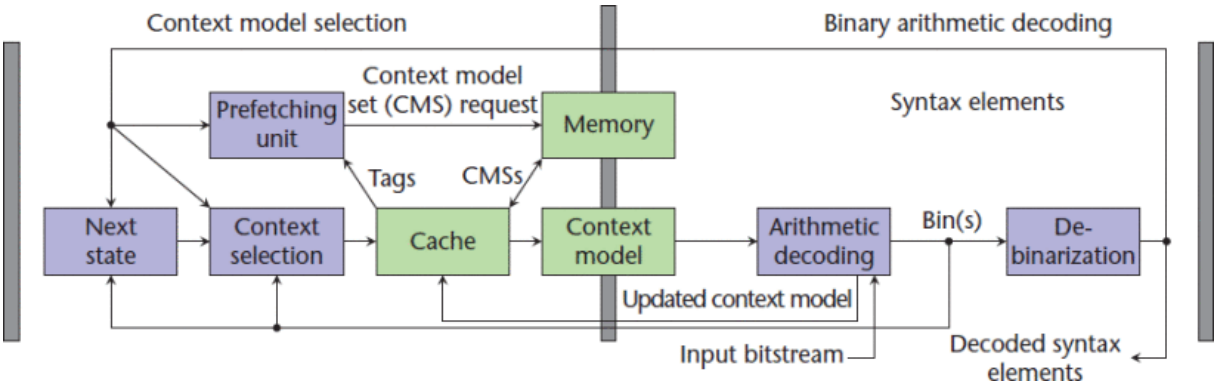

Figure 1. The proposed decoder architecture's two stage pipeline: context model selection and binary arithmetic decoding. The green modules represent synchronous memories, while the purple modules represent combinational logic.

Context Model Cache

We implemented a non-cached version of the decoder as a reference where the context model memory is directly accessed. The memory contains 16 memory sets of context models and is capable of fast in-memory copies. This allows the maintenance of multiple context model memory sets and thus supports efficient CABAC decoding when high-level parallelization tools are used (such as wavefront processing or tiles). A cache can be used to replace the context model memory in the critical path and thereby allow a higher clock frequency. The prefetching unit sends CMS requests to the memory. The CMSs are written back when they must be replaced. In our implementation, a cache miss results in a miss penalty of two clock cycles while the missing CMS is loaded from memory. CMS replacement is handled by a least recently used (LRU) policy. The cache is fully associative and contains a generic number of cache lines (CLs) ranging from 1 to 64, each storing one CMS. The decoder can be configured to use CMSs of four or eight context models.

Figure 2 shows the optimized context model memory layout for the configuration with eight context models per CMS (CMS8). Context models for the same type of syntax element are grouped primarily to exploit spatial locality; examples include last_sig_coeff_x/y_prefix (lines 0–4 in Figure 2), sig_coeff_flags (lines 8–10 and 12–13), and coeff_abs_level_greater1/2_flags (lines 16–21). In other cases, we grouped context models for consecutively decoded syntax elements (lines 5–7 and 14). Figure 3 shows an example where the decoding flow can go three different ways based on the decoding of two consecutive syntax elements. This can make the prefetching of the required context models very difficult, because it must be initiated two clock cycles before the context models are needed. The CMS8 configuration lets us remove the dependency from the decoded pred_mode_flag, as all potential next context models fit in a CMS (line 5), while there is no conflict with the goal to exploit spatial locality (part_mode requires four different context models). The memory layout for four context models per CMS (CMS4) cannot use this technique as often as CMS8. This is one of the advantages of using eight instead of four context models per CMS. Another advantage is the improved ability to exploit spatial locality, especially for the sig_coeff_flags of 4×4 transform blocks (lines 10 and 12) and the consecutively decoded last_sig_coeff_x/y_prefixes (lines 0–4), as the required corresponding context models exceed CMS4 capacity.

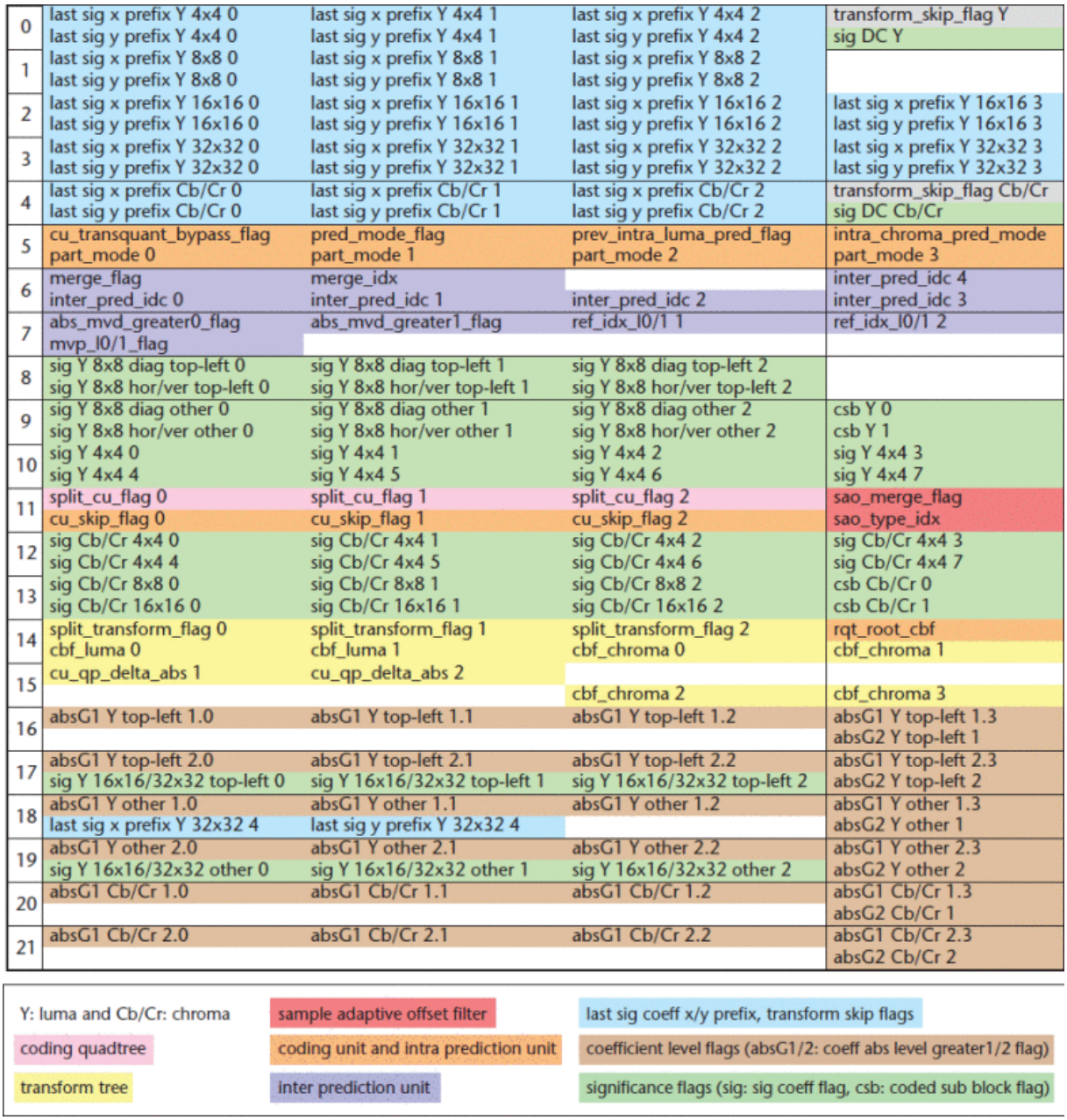

Figure 2. Context model memory layout for a configuration with eight context models per set (CMS8).

Other CMS sizes are also imaginable. However, at least three last_sig_coeff_x/y_prefix and sig_coeff_flag context models and four coeff_abs_level_greater1_flag context models are potentially needed to decode a 4×4 transform sub-block. As these sub-blocks contribute most of the bins in high-bitrate bitstreams, the CMS4 configuration already represents the smallest reasonable CMS size. Bigger CMS sizes require the merging of context models for different types of syntax elements to keep the memory overhead small. Unfortunately, the context models for decoding consecutive groups of equal syntax elements often depend on different parameters. For example, sig_coeff_flags depend on the transform block size and scan pattern, while coeff_abs_level_greater1_flags depend on the decoded bins in the previous 4×4 subblock. This limits the expected gain of bigger CMS sizes. Because we must consider

other issues – including hardware cost and memory bus width – we limit our evaluation to the CMS4 and CMS8 configurations.

Context Model Prefetching

Application-specific context model prefetching can significantly reduce the cache miss rate in HEVC CABAC decoding. Admittedly, the required context model often depends on the results of the previously decoded bin and cannot be prefetched early enough. However, most of the time, we can be sure that the required context model is contained in a specific set of context models. If this set is available in the cache, it is not necessary to know the exact context model in advance to guarantee a cache hit.

As Figure 4 shows, the prefetching module reads the control state machine's current state, the video sequence parameters (such as slice type and minimum coding unit size), some decoded syntax elements (such as prediction mode and prediction unit partition mode), and the currently decoded bin. Based on this information, it selects up to two CMS candidates that will likely be needed soon. Because the module tracks the CMSs in the cache, it can see if they are there. If one is not, it sends a read request to the context model memory. The first candidate has a higher priority than the second, so the second is fetched only when the first is already available. Unfortunately, this can lead to the first candidate being available but replaced by the second because it is the next to be replaced according to the LRU policy. To avoid this, we implemented a refresh mechanism that resets the first CMS candidate's LRU index if it is already in the cache so it will not be replaced next.

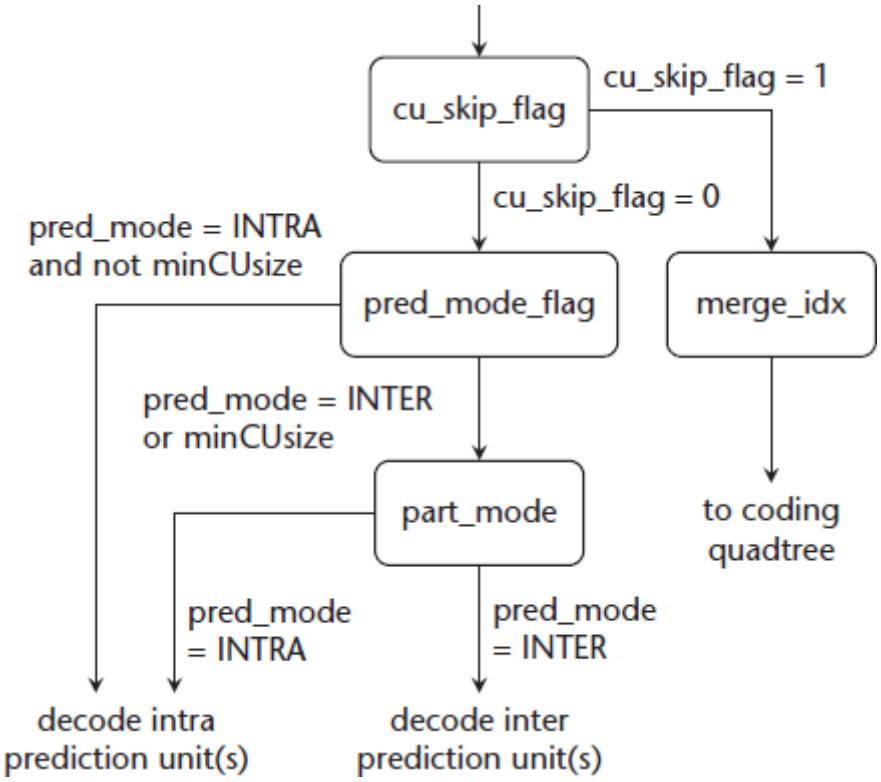

Figure 3. Coding unit (CU) decoding flow extract in a B/P-Slice. In this example, the decoding flow can go three different ways based on the decoding of two consecutive syntax elements.

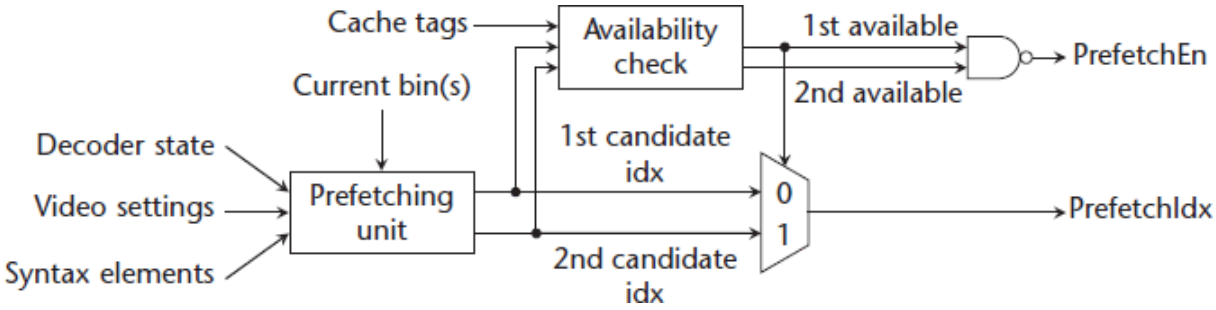

Figure 4. Overview of the prefetching candidate selection process. It reads the control state machine's current state, the video sequence parameters, some decoded syntax elements, and the currently decoded bin, then uses this information to choose up to two context model set (CMS) candidates that will likely be needed soon. The second candidate has a lower priority and is only selected if the first candidate is already in the cache.

The prefetching strategy depends on the number of available CLs. The strategy for at least four CLs fetches CMSs that might be used soon, while for smaller caches, CMSs are fetched only when there is a high probability that they will be needed. In general, the strategy for two CLs is more careful to avoid replacing CMSs that might still be used. Prefetching with one CL can be used only when no CMS is currently in use or if we know when the CMS will not be needed anymore. Also, in the one-CL strategy, the second candidate is not used.

Designing a general prefetching strategy is very challenging if it has to work well for all kinds of video sequences. Both encoding parameters and video content characteristics typically influence the decoding flow and must be taken into account. Although we can consider encoding parameters such as the quantization parameter, predicting video characteristics is not that obvious. We solved this problem by using context models for decisions in the prefetching strategy. Context models store an estimated probability that the corresponding syntax element will have a specific value. The MPS can be used to predict branches in the decoding flow with high accuracy. Because context models are adaptable to video characteristics, the prefetching strategy inherits this capability. Our proposed prefetching algorithm tracks the MPSs of the context models for four syntax elements: cu_skip_flag, pred_mode_flag, cbf_luma, and last_sig_coeff_y_prefix (only for first bin). Figure 3 offers an example of how this prediction works. Before entering the presented decoding flow, either the CMS containing the pred_mode_flag context model or the CMS containing the mergeidx context model is prefetched. The decision is based on the MPS of the cu_skip_flag context model. Basically, the CMS for the more probable branch is fetched. The same prediction is performed for the pred_mode_flag in the CMS4 configuration.

Evaluation

To validate the proposed CABAC hardware decoder's functionality, we used a hybrid HEVC decoder as hardware-software codesign on the Xilinx Zynq-7045 SoC. Our decoder is implemented in Zynq Programmable Logic – the Kintex-7 field-programmable gate array (FPGA); the remaining decoder components are executed by the ARM Cortex-A9 CPU. We used the highly optimized HEVC software decoder developed by the Embedded Systems Architecture Group at Technische Universität Berlin.[8] We used five test sequences from the

JCT-VC class B test set (1080p) in the evaluation.[9] The sequences are encoded in all-intra (AI) and random-access (RA) mode, with quantization parameters (QPs) of 17, 22, 27, 32, and 37. The QP of 17 is added to those specified in the JCT-VC Common Test Conditions to cover a wider range and get more meaningful results, especially for very high-bitrate videos.

In the following, we evaluate the proposed decoder for different cache configurations. Here, a configuration $M \times N$ represents a cache with M CLs and N context models per CL. We evaluated all configurations from 1×4 to 64×4 and from 1×8 to 32×8. The 64×8 configuration is not needed, because all required context models for decoding a coding tree unit (CTU) fit into 32 CLs due to the increased CL capacity.

Clock Frequency

The purpose of replacing the data path's context model memory with a smaller cache is to shorten the critical path and thereby increase the achievable clock frequency and throughput. The proposed design has been synthesized with Xilinx Synthesis Technology 14.6 (with the goal of optimizing speed). Both the memory and the cache are forced to be synthesized with the same FPGA resources to get a fair comparison that is valid beyond FPGAs. Figure 5 shows the influence of the number of CLs and the number of context models per CMS on the decoder's maximum clock frequency.

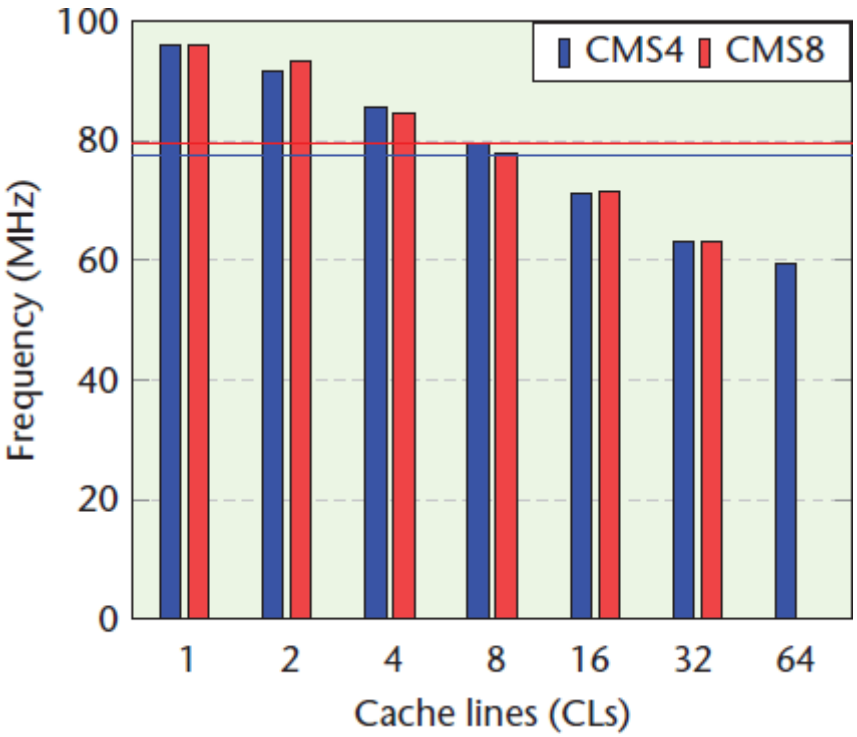

Figure 5. Decoder clock frequency for a different number of cache lines (CLs) and context models per CMS (CMS4/8). The horizontal lines show the clock frequencies for the corresponding non-cached designs.

First, we should note that no consistent clock frequency difference exists between the configuration with four and eight context models per CMS (CMS4/8). The small variations can be explained by the FPGA synthesis technology, which does not offer completely predictable results. The independence of the CMS size is surprising; the standard expectation is that bigger CMSs would result in a higher delay during the selection of the required context model from the CMS. However, when the CMS size is doubled from four to eight context models, only half of the CMSs are needed to contain all of the context models. This shortens the CMS address by one bit and thereby simplifies its computation. The simplification of the address generation logic compensates for the bigger selection tree and keeps the overall critical path delay constant.

On the other hand, the number of CLs significantly affects the achievable clock frequency. For CMS4/CMS8, respectively, clock frequency increases by 23.4/20.7 percent for a single CL; by 18.0/17.0 percent for two CLs; and by 10.1/6.1 percent for four CLs. We found only a marginal variation for eight CLs-2.6 percent for CMS4 and −2.0 percent for CMS8 – and the clock frequency is even reduced for bigger caches. The rapid clock frequency reduction comes from both the LRU implementation and the CL selection, which are not well suited for bigger caches with full associativity. Also, we should note that these results can vary for different implementations. For example, shorter pipeline stages can lead to greater relative improvement.

Although improved clock frequency accelerates the decoding of all bins, only context-coded bins can result in cache misses. The fraction of context-coded bins in the test sequences varies from 63 to 78 percent. However, because up to two bypass-coded bins can be decoded in parallel, the decoding time fraction for context-coded bins increases slightly (from 73 to 84 percent).

Performance without Prefetching

Unfortunately, improved clock frequency comes at the cost of potential cache misses. These, in turn, lead to stalls in the decoder pipeline and thereby reduce the overall throughput.

Figure 6 shows the cache miss rate for different cache sizes and video modes for CMS4 (Figure 6a) and CMS8 (Figure 6b) configurations. The results show the arithmetic mean of the five different test sequences. In general, the miss rate grows with higher QPs, because smaller QPs result in more bins due to less quantization. Because bins of the same syntax elements are grouped, temporal and spatial locality can be better exploited when accessing the required context models in the cache. Our results show a significant miss-rate reduction for all video modes when the number of CLs is increasing. However, there is no noticeable improvement with 64×4 and 32×8 configurations. Here, all required context models for decoding a specific CTU fit in the cache, and all resulting cache misses are cold misses during the first access. In this case, the fraction of missing accesses decreases with the total number of context-coded bins, which is higher for smaller QPs. It's rare that all the context models would be used during CTU decoding, especially if only a few bins are decoded, as in the RA QP37 configuration. In such cases, 32×4 and 16×8 are also sufficient and lead to the same results as 64×4 and 32×8.

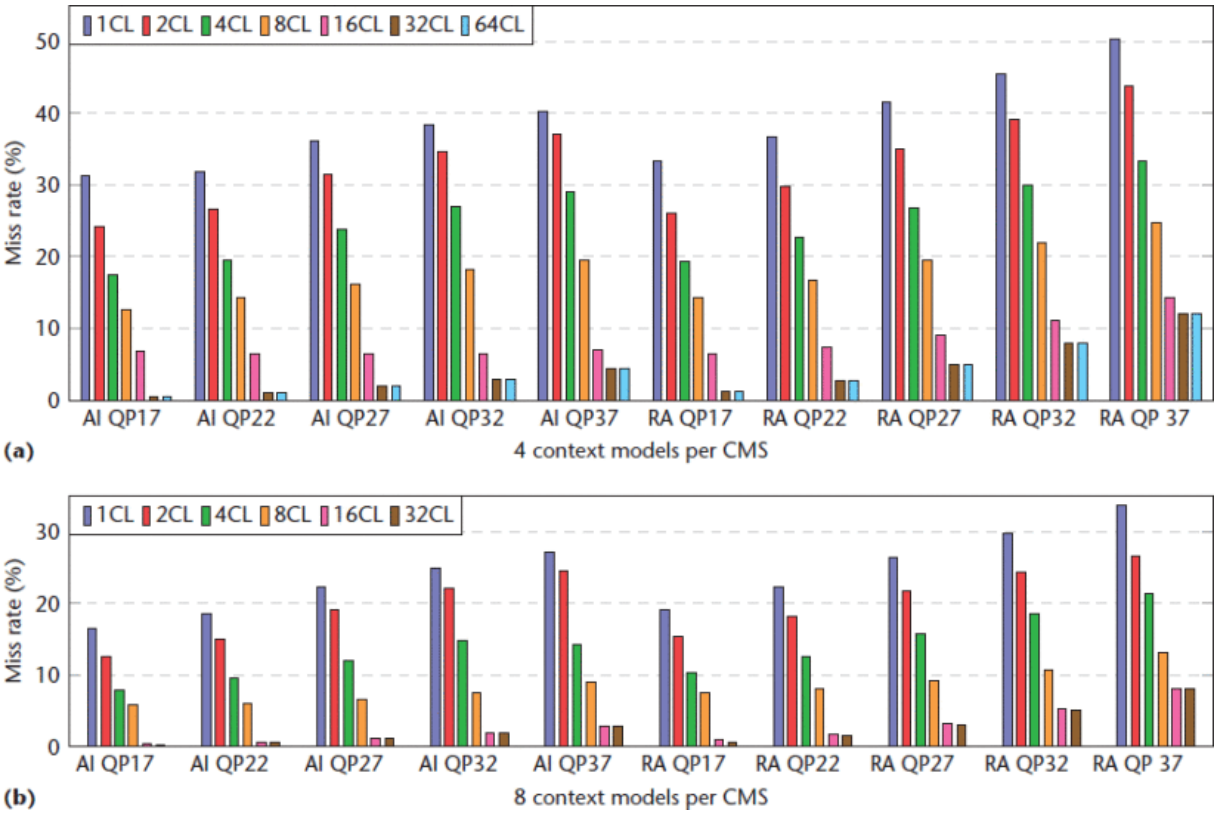

Figure 6. Cache miss rate without prefetching, The results show the arithmetic mean of the five different test sequences for sets of (a) Four (CMS4) and (b) Eight (CMS8) context models.

Because fewer than eight CLs results in high miss rates and more than eight CLs produces reduced clock frequencies, an overall performance improvement cannot the reached without prefetching. Even in the best case – with an AI QP17 video and a 2×8 cache, which results in a 73 percent context-coded bin cycle fraction, 17 percent higher clock frequency, and 11 percent miss rate – the overall performance is the same as with a non-cached decoder.

Performance with Prefetching

The proposed prefetching algorithm significantly reduces the cache miss rate. We show the miss rates with only one CL separately in Table 1, because they are much higher than when more CLs are used.

A 1×4 cache is still not acceptable, because the miss rate is greater than 15 percent for all videos because of the restricted prefetching opportunities. As Figure 7 shows, the 1×8 configuration allows a satisfying overall throughput improvement of 7.0-13.1 percent, but only for high-bitrate videos (AI QP17 and QP22, and RA QP17). For lowest bitrates, the decoder performance is up to 11.5 percent less than with the non-cached baseline. Unexpected results can be observed for the 1×4 configuration, where AI QP17 videos lead to a higher miss rate than AI QP22 videos. Usually, the miss rate decreases with a higher bitrate due to the improved exploitation of temporal and spatial locality. However, the fraction of 4×4 transform blocks also increases with higher bitrates (from 64.5 percent for AI QP22 to 72.8 percent for AI QP17 on average). Unlike bigger transform blocks, these blocks require a

different context model for every bin of the last_sig_coeff_x/y_prefix syntax elements. Furthermore, nine instead of four context models are potentially needed for the sig_coeff_flags. These requirements, combined with the limited cache size of only four context models, lead to an increased miss rate.

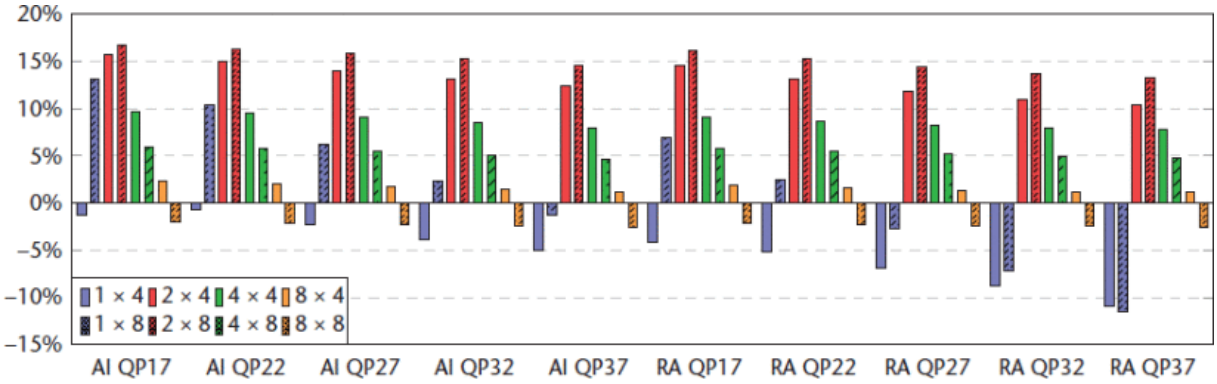

Figure 7. Throughput improvement with prefetching for different cache configurations (compared to non-cached decoder).

Table 1. Cache miss rates (in percent) with prefetching and one cache line for four and eight context models per set (CMS4/CMS8), all-intra (AI) and random-access (RA) modes, and quantization parameters (QP) from 17–37.

		QP17	QP22	QP27	QP32	QP37
CMS4	**AI**	16.40	15.85	16.95	17.78	18.32
	RA	17.31	18.29	19.98	21.55	23.24
CMS8	**AI**	4.39	6.09	8.75	11.23	13.54
	RA	7.73	10.84	14.81	18.35	21.90

Figure 8 shows the miss rates for more than one CL. A 2×4 cache already significantly improves the miss rates for all video configurations. As Figure 7 shows, miss rates of 1.3-4.2 percent lead to overall throughput improvements of 15.7-10.4 percent, due to the 18 percent improvement in clock frequency. With a 4×4 cache and all configurations with more CLs, the miss rate is less than 1.3 percent for all videos. As a result, the decoder performance is no longer noticeably affected, while almost all of the clock frequency improvement remains. The 4×4 cache results in a consistent throughput gain of 7.8-9.7 percent. More CLs lead to even smaller miss rates; however, a significant speed-up cannot be achieved because of the constant or decreased clock frequency.

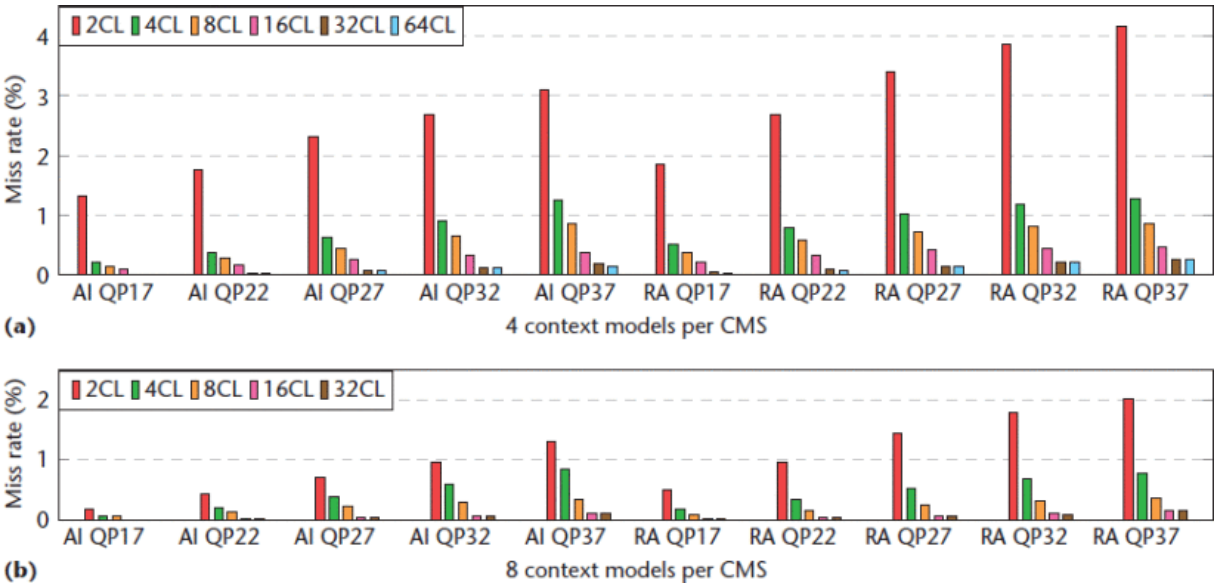

Figure 8. Cache miss rate with prefetching.

The resulting miss rates for all configurations with CMS8 are qualitatively similar to the ones with CMS4, but slightly lower. For a 2×8 cache, the miss rates range from 0.2 percent to 2.0 percent, leading to an overall performance improvement of 16.7 to 13.2 percent, respectively. More CLs reduce the miss rates to less than 0.9 percent and make them negligible. The 4×8 configuration can increase the throughput by 4.8-6.0 percent, which is mainly limited by the clock frequency improvement of 6.1 percent. More CLs reduce the decoder performance, because the clock frequency falls below that of the non-cached baseline.

Figure 7 compares the throughput improvement for all cache configurations with one to eight CLs; as the figure shows, two CLs lead to the highest improvements. Although the 2×4 cache allows for a slightly higher clock frequency improvement – 18 percent, compared to 17 percent for the 2×8 cache – the latter reaches the greatest overall performance gain because the miss rates are less than half due to the doubled CMS size. For a single CL, the throughput varies heavily as the miss rate strongly depends on the video characteristics. In contrast, for four and eight CLs, the throughput improvements are similar for all videos: all miss rates are less than 1.3 percent, which allows a throughput gain that is close to the clock frequency improvement.

Table 2. Comparison with other hardware decoders.

	Hardware decoders				
	Proposed	Yi5	Yang6	Hong7	Chen10
Standard	HEVC	H.264	H.264	H.264	HEVC
Technology	28 nm FPGA	180 nm CMOS	180 nm CMOS	130 nm CMOS	45 nm SOI*
Cache	2 × 8	1 × 8	2 × 9	2 × ?	—
Prefetching	×			×	
Frequency (Mhz)	93	225	140	333	1,600
Bins/cycle	1.13	0.24	0.86	1.08	1.06
Throughput (Mbins/s)	106	54	120	360	1,696

*SOI = silicon-on-insulator (SOI) process technology.

We can conclude that a 2×8 cache allows the highest speed-up for all test videos. Although it contains two times the number of context models as the 2×4 cache, the size is still very small (2CLs×8context models×7bits=112bits). Table 2 shows a comparison to previous cached CABAC hardware decoders. We can compare our HEVC CABAC decoder to H.264 CABAC decoders because the general CABAC algorithm is the same in both standards. However, the throughput might be slightly increased in HEVC CABAC due to some optimizations in the standard that allow the decoding of more bins per clock cycle. We also present Yu-Hsin Chen and Vivienne Sze's decoder, as it represents the state of the art. Our configuration is very similar to that of Yao-Chang Yang and Jiun-In Guo,[6] who used a 2×9 cache for an H.264 CABAC decoder. Yu Hong and colleagues also implemented a cache with two CLs,[7] but did not provide information about the CL size. Yongseok Yi and In-Cheol Park's proposal[5] used only one CL of eight context models, which might reduce the overall performance due to the mandatory stall cycles for each CL switch.

Our decoder has throughput similar to that of 180 nanometer (nm) CMOS decoders, but it is outperformed by the 130 nm CMOS implementation due to the significantly higher clock frequency. Although our decoder is implemented using an advanced 28 nm FPGA, it cannot reach the competitors clock frequencies due to the inherent routing logic overhead and the customization limits of FPGA technology. Further, we cannot offer a more detailed comparison of the cache architectures, because neither the clock frequency increase nor the resulting miss rates are presented in the former works.[5–7] Chen and Sze's HEVC decoder[10] provides a significantly higher throughput than all other competitors. The reason is the very high clock frequency of 1.6 GHz, which can be reached due to the five-stage pipeline and the advanced 45 nm silicon-on-insulator (SOI) process technology.

Real-Time Decoding

Table 3 and 4 show the real-time decoding capabilities of the proposed HEVC CABAC decoder with a 2×8 cache and prefetching. Table 3 shows the average throughput in RA test videos with one intra frame per second. The throughput requirements are fulfilled for all videos with a QP of 22 or higher, but not for all QP17 videos. Nevertheless, a QP of 22

already allows for high-perceptive video quality. As a consequence, the proposed HEVC CABAC decoder enables real-time decoding of high-quality Full HD videos with a low-power platform such as the Zynq-7045.

Unfortunately, significant variations exist in the required bitrate for different frames of the same video. Intra frames in particular often require much higher bitrates, as they cannot remove temporally redundant information in consecutive frames using inter-picture prediction. Table 4 shows the throughput requirements for videos that are encoded only with intra frames, indicating the peak throughput required to decode single frames in real time. However, because the varying throughput requirements also depend on a frame's content, the real peak throughput will be higher. The requirements are met for all videos with a QP of 27 or higher and for some QP22 videos. This means that the proposed CABAC decoder cannot fulfill the real-time requirements for every frame separately. However, if the decoding latency of a few frames and the additional hardware for frame buffers are acceptable, a judder-free streaming can be realized.

Table 3. Average throughput (Mbins/s) for random-access mode.

	Quantization parameters				
Video* (1080p)	QP17	QP22	QP27	QP32	QP37
BasketballDrive (50 fps)	97.7	95.6	92.9	90.2	87.6
	88.6	22.1	7.5	3.5	1.9
BQTerrace (60 fps)	98.5	94.2	91.0	88.1	84.9
	175.7	57.2	10.2	3.1	1.4
Cactus (50 fps)	97.1	94.2	93.0	90.0	86.9
	101.8	23.5	6.9	3.3	1.7
Kimono (24 fps)	98.1	98.3	96.1	93.2	89.6
	17.2	5.7	2.6	1.3	0.7
ParkScene (24 fps)	96.3	94.1	91.9	89.1	86.1
	24.5	9.2	4.0	1.9	0.9

*The first line for each video shows the achieved throughput with a 2 x 8 cache and prefetching, while the second line shows the required throughput for real-time decoding. The entries in bold type indicate that real-time throughput requirements have not been met.

Table 4. Average throughput (Mbins/s) for the all-intra mode.

	Quantization parameters				
Video* (1080p)	QP17	QP22	QP27	QP32	QP37
BasketballDrive (50 fps)	101.4	99.0	95.8	92.4	89.4
	219.9	89.4	36.7	20.1	11.7
BQTerrace (60 fps)	105.6	102.6	99.6	95.0	91.1
	331.8	224.9	95.8	49.6	28.0
Cactus (50 fps)	103.9	101.0	98.6	94.8	91.3
	285.0	127.2	58.2	32.2	17.9
Kimono (24 fps)	103.1	105.1	102.6	99.2	95.7
	66.0	26.1	14.4	8.3	4.8
ParkScene (24 fps)	105.3	102.9	99.2	95.3	91.9
	117.3	62.3	34.3	18.3	9.3

*The first line for each video shows the achieved throughput with a 2 x 8 cache and prefetching, while the second line shows the required throughput for real-time decoding. The entries in bold type indicate that real-time throughput requirements have not been met.

Resource Utilization

Table 5 compares the resource utilization of the non-cached CABAC decoder using CMS4 and CMS8 with the corresponding cached designs with two CLs. To compare various designs, we used area optimization to perform synthesis and achieve meaningful results. We also show results with speed optimization to allow a fair comparison with other implementations.

Table 5. Resource utilization on the Xilinx Zynq-7045 SoC.

Design*	Registers	Look-up table (LUT)	Block RAM (BRAM)	Digital Signal Processor (DSP)
Base CMS4 (area optimized)	3,008	7,562	15	1
	0.23%	1.15%	2.75%	0.11%
2 3 4 cache (area optimized)	3,193	8,157	15	1
	0.24 %	1.24%	2.75%	0.11%
2 x 4 cache (speed optimized)	3,239	9,046	15	1
	0.25%	1.38%	2.75%	0.11%
Base CMS8 (area optimized)	3,061	7,628	15	1
	0.23%	1.16%	2.75%	0.11%
2 x 8 cache (area optimized)	3,352	8,2	15	1
	0.26%	1.25%	2.75%	0.11%
2 x 8 cache (speed optimized)	3,402	9,152	15	1
	0.26%	1.40%	2.75%	0.11%
Total available	1,311,600	655,8	545	900

*Each cell shows the absolute (first line) and relative (second line) device utilization.

We can draw three main conclusions from the results. First, the cached designs (2×4 and 2×8) require only 6.2 and 9.5 percent more registers and 7.9 and 7.5 percent more look-up

tables (LUTs), respectively, compared to the non-cached designs. This is a reasonable trade off, considering that the decoder throughput is increased from 10.4 to 16.7 percent. Second, the decoder with a 2×8 cache utilizes 5.0 percent more registers and 0.5 percent more LUTs than the decoder with a 2×4 cache, while offering 1.8 percent more in average throughput improvement. However, because the register utilization is only a quarter of a percent, it is not critical and the design with the higher throughput might be a better choice. Finally, less than 3 percent of the FPGA resources are needed to implement the CABAC decoder, including the processor interface. This means that more decoder components can be implemented in the FPGA, as most of its resources remain available for use.

The number of block memories (BRAM) is constant for all configurations. However, only one BRAM is used as the context model memory for the actual decoder. The remaining BRAMs are required to store all decoded syntax elements. A single digital signal processing unit (DSP) is used in all configurations to perform the inverse quantization of decoded transform coefficients.

A comparison to related work is difficult; all previous cached CABAC hardware decoders were implemented in 130 or 180 nm CMOS technology, while the proposed decoder is realized with an FPGA. Also, our decoder covers HEVC, while the others address its predecessor H.264.

Energy Efficiency

We measured the HEVC CABAC decoder's power consumption using the Xilinx Vivado Power Analysis tool. A constant static power of 241 mW is consumed by all designs. Because this static power is used only to keep the state of the configurable FPGA logic, we do not consider it in the evaluation. The dynamic power is of more interest, as it varies for the different designs. The best performing design has a 2×8 cache and consumes 88.3 mW.

Varying cache sizes affect the power consumption in different ways. On the one hand, a bigger cache requires more hardware resources, which consume more power. On the other hand, the miss-rate reduction that comes with bigger caches leads to less memory accesses, which saves power. Furthermore, as Figure 5 shows, the achievable clock frequency decreases significantly for bigger caches, as does the power consumption of the corresponding decoder. These opposing effects result in a power consumption that is almost constant for different cache sizes, as well as for the non-cached decoder. Consequently, the average throughput for various test videos dictates the energy efficiency (see Figure 9). A peak can be observed for both configurations (CMS4 and CMS8) with four CLs at approximately 1.1Gbins/Joule. These designs are 8.5 and 12.0 percent more energy efficient, respectively, than the non-cached baselines. Although the throughput is higher with two CLs, this is achieved at a higher clock frequency, which degrades energy efficiency. More CLs also result in reduced energy efficiency, because the throughput is significantly lower.

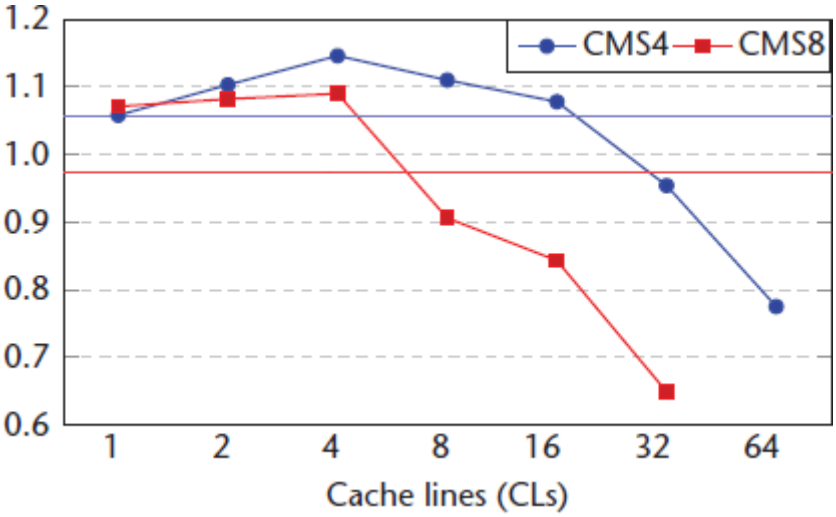

Figure 9. Energy efficiency in Gbins/Joule for various numbers of cache lines (CLs) and context models per CMS (CMS4 and CMS8). The horizontal lines show the energy efficiency for the corresponding non-cached designs.

Our results show that two CLs are sufficient to reduce the miss rate to the point where it only marginally affects the overall performance, while maintaining nearly the full clock frequency improvement. The decoder configurations with two CLs are most promising, because they allow the highest speed-ups for all test videos compared to a non-cached decoder. The 2×4 cache reduces the miss rate to a fraction of 1.3 percent up to 4.2 percent, which leads to a speed-up of 15.7 and 10.4 percent, respectively, thanks to the 18 percent clock frequency improvement. Although the 2×8 cache's clock frequency improvement is slightly smaller (17 percent), its overall throughput gain ranges from 16.7 to 13.2 percent, due to the miss rates of 0.2 percent to 2.0 percent, respectively. Four CLs allow for even lower miss rates, but the overall speed-up is limited due to the clock frequency improvement of only 10.1 percent for a 4×4 cache and 6.1 percent for a 4×8 cache. A single CL results in miss rates of more than 20 percent and leads to significant throughput degradations for low bitrate videos.

Our best-performing configuration (a 2×8 cache) makes real-time decoding of high-quality Full HD videos on a low-power platform such as the Zynq-7045 possible. This cached decoder is 12 percent more energy efficient on average than the non-cached decoder.

Despite the direct throughput improvement from enhanced clock frequency, other designs might remove the pipeline stage that performs the context model memory access when the cache can be shifted to an adjacent stage. This shortened pipeline might also significantly improve the throughput as the strong dependencies in CABAC decoding make deep pipelining inefficient.

Acknowledgment

This work received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 645500 (Film265).

References

1. G.J. Sullivan et al., “Overview of the High Efficiency Video Coding (HEVC) Standard,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1649–1668.

2. T.Wiegand et al., “Overview of the H.264/AVC Video Coding Standard,” IEEE Trans. Circuits and Systems for Video Technology, vol. 13, no. 7, 2003, pp. 560–576.

3. D. Marpe, H. Schwarz, and T.Wiegand, “Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard,” IEEE Trans. Circuits and Systems for Video Technology, vol. 13, no. 7, 2003, pp. 620–636.

4. V. Sze and M. Budagavi, “High Throughput CABAC Entropy Coding in HEVC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1778–1791.

5. Y. Yi and I.-C. Park, “High-Speed H.264/AVC CABAC Decoding,” IEEE Trans. Circuits and Systems for Video Technology, vol. 17, no. 4, 2007, pp. 490–494.

6. Y.-C. Yang and J.-I. Guo, “High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications,” IEEE Trans. Circuits and Systems for Video Technology, vol. 19, no. 9, 2009, pp. 1395–1399.

7. Y. Hong et al., “A 360 Mbin/s CABAC Decoder for H.264/AVC Level 5.1 Applications,” Proc. IEEE Int’l SoC Design Conf. (ISOCC), 2009, pp. 71–74.

8. C.C. Chi et al., “SIMD Acceleration for HEVC Decoding,” IEEE Trans. Circuits and Systems for Video Technology, vol. 25, no. 5, 2015, p. 841–855.

9. F. Bossen, Common HM Test Conditions and Software Reference Configurations, JCTVC-L1100, Joint Collaborative Team on Video Coding, 2013.

10. Y.-H. Chen and V. Sze, “A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-tier Applications,” IEEE Trans. Circuits and Systems for Video Technology, vol. 25, no. 5, 2015, p. 856–868.

About the Authors

Philipp Habermann

Technische Universität Berlin

Philipp Habermann is a doctoral student and research assistant in the Embedded Systems Architecture group at Technische Universität Berlin (TU Berlin). His research interests include high-performance computer architectures and video coding, especially context-based adaptive binary arithmetic coding. Habermann received an MS in computer engineering from TU Berlin.

Chi Ching Chi

Technische Universität Berlin

Chi Ching Chi is a doctoral student in the Embedded Systems Architecture group at Technische Universität Berlin and chief technology officer at Spin Digital Video Technologies GmbH, which he cofounded in 2015. His research interests include video coding, video acceleration, and scalable video systems. Chi received an MS in computer engineering from Delft University of Technology, the Netherlands.

Mauricio Alvarez-Mesa

Technische Universität Berlin

Mauricio Alvarez-Mesa is a senior researcher at Technische Universität Berlin and CEO of Spin Digital Video Technologies GmbH. His research interests focus on high-performance video coding. Alvarez-Mesa received a PhD in computer engineering from Polytechnic University of Catalonia.

Ben Juurlink

Technische Universität Berlin

Ben Juurlink is a professor in the Embedded Systems Architecture group at Technische Universität Berlin. His research interests include multi- and many-core processors, reconfigurable computing, and the art of mapping applications effectively and efficiently to computer architectures. Juurlink received a PhD in computer science from Leiden University, The Netherlands